Building A Music Recommendation Engine Using PageRank & Naive Bayes Classifiers

Omar Hegazy

Abstract

Music recommendation engines, such as Spotify or Pandora, are currently built through a variety of methods depending on the service. Pandora employs a manual discovery model in which musicologists take 20 to 30 minutes to assign weight values for 450 musical attributes for every song on the service. While The Music Genome Project has a wide array of songs, this algorithm by it's nature cannot scale as fast as an automated service -- It took Pandora over 10 years to accumulate 1 million songs. By comparison, a well-designed automated service can analyze over 10 million artists in a much shorter amount of time.

iTunes Genius, Amazon, and Last.fm employ collaborative filtering models in which customers with a few common interests are used for recommendation sources between each other.This algorithm makes the naive assumption that just because 2 customers have one common interest, they will also have many other common interests. Furthermore, it's focus on existing user data can lead to perpetuating already known interests.

Acoustic analysis engines such as Muffin often turn a blind eye to the cultural aspects of music and have trouble boiling down musical tastes to statistical analyses of signals.

A newer approach is to analyze and develop an artist network, focusing purely on online articles about artists. By seeing which of these articles link to each other, and which artist articles have the highest amount and most popular incoming links, the algorithm can determine which artists are most related and which artists customers will have the highest probability of liking. Unlike all other mentioned models, it is automated, focuses on the community of artists as opposed to user data, and looks at a corpus of text written about the artists instead of analyzing audio files.

Introduction

My algorithm starts by scraping an initial dataset for artist articles. In its first implementation, it uses Wikipedia as it's database. Despite the fact that Wikipedia has a painfully small amount of artist articles (only estimated 130,000 total), Wikipedia is the easiest database of artists I could find to scrape. After finding patterns in articles to locate artists, the algorithm begins cleaning up incorrect elements in the artist list. Finally, the algorithm determines the relatedness between artists and assigns a popularity score to each artist dependent on his/her volume of incoming links and the popularity of those links. Then, I will determine the success of that method by comparing it to the artists' iTunes sales records and number of Spotify plays. If the method is successful, the final program will take an artist as user input and return related artists, sorted by the the more successful method of determining probability.

Preliminary Information

Bayes' Theorem

Bayes' theorem is an important statistical rule in determining conditional probabilities. It is a core element in my algorithm for distinguishing the difference between artist articles and random articles, to clean out non-artist articles that my scraper accidentally added to the dataset. Determining the probability of event **A** given the occurrence of event **B** is expressed as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Proving Bayes' Theorem can be simply done using the axioms of conditional probability :

$$P(A|B)P(B) = P(A) \cap P(B)$$
$$P(B|A)P(A) = P(A) \cap P(B)$$

Then, it is a matter of simplification to get to our result :

$$P(A|B)P(B) = P(B|A)P(A)$$
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

If we can assume that the event space of A is binomial (only A and not A can occur) , the denominator can be rewritten as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}$$

If the event space of B is concerned with numerous possible events, we can calculate combined probabilities as follows:

$$P(A|B) = \frac{\prod P(B_j|A) * P(A)}{\prod P(B_j|A) * P(A) + \prod P(B_j|\neg A) * P(\neg A)}$$

This makes the assumption that $P(B_1$ and B_2 and B_3 …$|A)$ is equal to $\prod P(B_j|A)$, which is only true if each of the events in B are independent of each other.

In the event that we want to express this formula in terms of $P(A|B_j)$ instead of $P(B_j|A)$, we can apply Bayes' Theorem on $P(A|B_j)$, plug in the result and simplify (see Reference 3 for full application of arithmetic) to get to this expression :

$$P(A|B) = \frac{\frac{\prod P(A|B_j)}{P(A)}}{\frac{\prod P(A|B_j)}{P(A)} + \frac{\prod P(\neg A|B_j)}{P(\neg A)}}$$

If we make the naive assumption that P(A) = P(~A), then we come to this final expression for combined conditional probabilities:

$$P(A|B) = \frac{\prod P(A|B_j)}{\prod P(A|B_j) + \prod P(\neg A|B_j)}$$

Assuming P(A) is always 0.5 might seem strange, but in the application of our algorithm, all it means is that we must have equal amounts of artist articles and random articles in our testing. P(A) will represent the probability of an article being an artist article, and B will represent a given article's set of words.

Building the Program

Brief Introduction to the MediaWiki API

I use the Python requests library to obtain data from web page and the MediaWiki API allows me to obtain Wikipedia article data using HTTP requests. For an example, if I wanted to download the contents of an article titled "Wikipedia", I would run the following Python code :

```
header = "https://en.wikipedia.org/w/api.php?"
#required header for all API requests
footer = "action=query&titles='Wikipedia'&prop=revisions&rvprop=content"
"""the type of action is required, most of the time its 'query'. The ampersands signify optional
parameters. "&prop=revisions&rvprop=content" allows me to scrape article data from the most
recent revision."""
page = header + footer
page_data = requests.get(page).text.encode("utf8")
#sends http requests for the "page" webpage, and converts the result into unicode text
```

The API also allows me to get information from specific sections in an article with the "`&section=`" parameter, which takes in the integer number of the section you want. If I wanted to download information from an article section by it's name, I would have to download section metadata in XML form

with "`&prop=sections`" instead of "`&prop=revisions&rvprop=content`". This contains information for the relationship between section titles and section numbers. My scraper will look up the name of the requested section in the metadata and then return the number that section was assigned. This final number is then plugged into the "`&section=`" parameter. For more information, see Reference 5 and 6, as well as the code attached.

Analysis of the Scraping

I start by scraping the "Lists of musicians" article, which contains a list of links to genre articles. Each genre article contains a list of links to artists, which we scrape for artist names. Since it is unfeasible to run my program on every single artist from every single genre article, each time I test my program I take a random subset of the list of genres and analyze only the artists contained in that list. The subset is of arbitrary size; I usually take anywhere from 30 to 50 genre lists depending on how strong the network is where I'm testing, or even 200 if I can let my computer run it overnight.

After determining the genres the program will analyze, I begin scraping for artist names in the genre articles. The scraper first separates each article into sections and determines which sections have lists of artists, and which don't. Most of the genre articles are similarly formatted such that the list of artists are broken up alphanumerically into sections of letter and number ranges. My scraper can successfully assume that section titles like "0-9", "A", and "Aa-An" contain sections with lists of artists, while sections titles like "References" or "See also" contain sections with unrelated links. Even though we will use Bayesian reasoning to prune out unrelated links later in the program, it is important that the links we scrape in this early stage at least have a majority of artist articles.

Again, there are exceptions to the rule, as certain genre articles contain their list in uniquely titled sections. For example, the "List of singer-songwriter" genre article sorts it's list of artists by country,

rather than by ranges of numbers or letters. Every section in this page is reported as a ‘bad’ section and the entire page is ignored. This is quite a massive article to consistently ignore, and the sections actually possess only artist links.

To alleviate this problem, every time my scraper recognizes an article with an inordinate amount of ‘bad’ sections, it prints out the names of the bad sections and the name of the article containing them. Currently my scraper reports an error if there are 4 or more detected ‘bad’ sections in a row.

Then I have to manually look up the article, see if it is uniquely formatted (some just coincidentally have bad section titles while still putting their artist data in uniformly formatted sections), and then determine if it is worth the time to write scraping code specifically for that article. Some articles can be ignored because they have too non-uniform formatting, but some can be selectively accepted due to non-uniform formatting that can still be comprehended by the script.

After retrieving section information, I add all links to other Wikipedia articles in my database of artists. This makes the naive assumption that all links in the genre articles are to artists only, which, as I said, is true most of the time (the minority can be cleaned with further Bayesian reasoning). I get rid of any markup text and cleaned up specific formatting issues (for example, replacing “&” with “&”), and add all new artist names to the existing dataset.

Contents

0–9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

0–9 [edit]

- +44[1]
- !!!
- 3 Colours Red
- 3 Doors Down
- 10,000 Maniacs
- 13 Engines
- 1990s
- 311
- 4 Non Blondes
- 54-40
- 78violet
- 7 Year Bitch[2]
- 12 Stones[3]
- The 1975[4]

A [edit]

Above: normally formatted article (Alternative rock artists)

Another Animal	2006 – present	Pittsfield, New Hampshire	• *Another Animal* (2007)
Aranda	2001 – present	Oklahoma City, Oklahoma	• *Aranda* (2008) • *Stop the World* (2012) • *Not the Same* (2015)
Army of Anyone	2005–2007	Los Angeles, California	• *Army of Anyone* (2006)
Art of Dying	2005 – present	Vancouver, British Columbia	• *Art of Dying* (2006) • *Vices and Virtues* (2011)
Atom Smash	2006 – present	Miami, Florida	• *Love Is in the Missile* (2010) • *Beautiful Alien* (2012) • *Passage to the Sun* (2013)
Audioslave	2001–2007	Glendale, California	• *Audioslave* (2002) • *Out of Exile* (2005) • *Revelations* (2006)
Axium	1999–2006	Kansas City, Missouri	• *Matter of Time* (2002) • *Blindsided* (2003) • *The Story Thus Far* (2004)

Left : a uniquely formatted article I ignored (List of post-grunge bands)

B [edit]

Band	Years active	Origin	Studio albums
Ben Moody	1995 – present	Little Rock, Arkansas	• *All for This* (2009) • *You Can't Regret What You Don't Remember* (2011)

Chile [edit]

- Beto Cuevas
- Gepe
- Víctor Jara
- Javiera Mena
- Ángel Parra
- Javiera Parra
- Nicanor Parra
- Violeta Parra

Colombia [edit]

- Alci Acosta
- Albalucía Ángel
- Lucas Arnau
- J Balvin
- Naty Botero
- Cabas

Left: A uniquely formatted article I could not ignore (Singer-songwriters).

```
Finished section title 'Traditional singer-songwriters' from List_of_singer-songwriters
Finished section title 'Argentina' from List_of_singer-songwriters
Finished section title 'Australia' from List_of_singer-songwriters
Finished section title 'Austria' from List_of_singer-songwriters
Finished section title 'Barbados' from List_of_singer-songwriters
Finished section title 'Belgium' from List_of_singer-songwriters
Finished section title 'Bosnia and Hercegovina' from List_of_singer-songwriters
Finished section title 'Brazil' from List_of_singer-songwriters
Finished section title 'Bulgaria' from List_of_singer-songwriters
Finished section title 'Cambodia' from List_of_singer-songwriters
Finished section title 'Canada' from List_of_singer-songwriters
```

Above: Terminal outputting successfully scraped data from a uniquely formatted article. Notice the section titles aren't letter/number ranges, but my scraper can still retrieve the data.

Cleaning The Dataset

As I mentioned, all link data is accepted into the database. As you can see in the "Alternative rock artists" screenshot, a lot of the data in uniformly formatted articles are links to artists. However, to deal with the minority, I started by manually looking over my dataset after numerous runs of the program and noticing patterns of incorrectly scraped data. For example, I noticed a lot of the incorrect links were Wikipedia metadata, like "Image:[image_name.jpg]" or "Category:[Wikipedia category]". Other commonly scraped mistakes were intuitively blatant, like "[Album name] (album)". After writing code to detect and eliminate common mistakes like this, 99% of my dataset was clean.

The other 1% was much harder to eliminate. They were links to things like "Violinists", "England", or "The Guardian" (a British newspaper). These mistakes had no patterns between them, and unlike the metadata, they couldn't be identified by their textual formatting. To get rid of these errors, I had to build a naive Bayesian classifier to find the most common words that are used in both artist and non-artist articles to determine the "language" of each article type. Then, for each supposed artist in my dataset, I would determine the probability that it is a random article given it's wordlist compared to the common languages of artists and random articles.

The algorithm used for determining and analyzing the language of artist and random articles heavily uses Bayes' Theorem and the naive formula for conditional probabilities.

First I build a hash table of words from a list of perfectly identified artist articles, where words are mapped onto the amount of time those words show up in artist articles. To build this corpus, I use data from the artist articles I already scraped. Even though these artists aren't perfectly identified, 99% of them are, which means that the words in the 99% will affect the common language of artists far more than the wrong 1% and the final outcome probabilities will not be changed.

Then, I build a corpus of words from non-artist articles, with the same mapping of words onto amount of time those words show up in random articles. To build this corpus, I use the MediaWiki API to request random articles from Wikipedia. Even though there's a possibility that these random articles contain artist articles, again, 99% of the articles are not going to be artist articles, so the final probabilities of the wordlists will not be affected by the 1%.

A third hashtable is then built, where words are mapped onto the probability an article containing that word is a non-artist "random" article. To determine the probability that an article containing a given word is a non-artist article, we first need to determine the probability that a word exists in a given random article :

$$P(word|random) = 2.75\frac{R_A}{N_A}$$

Where $\mathbf{R_A}$ is the amount of times that word shows up in random articles, and $\mathbf{N_A}$ is the total amount of random articles. I use the number of random articles instead of the number of random words in the corpus because artist articles have far more words than random articles on average, but we analyze an equal amount of artist articles and random articles. Because random articles have far fewer common words and far less central of a language than artist articles, P(word|random) on words from random articles is usually lower than P(artist|random) on words from artist articles. To alleviate this bias, I multiply all computations of P(word|random) by the (arbitrarily selected, but effective) factor 2.75.

The probability that, given an artist article, a word will show up in it is similarly shown :

$$P(word|artist) = \frac{A_A}{N_A}$$

$\mathbf{A_A}$ is the amount of time that word shows up in artist articles, $\mathbf{N_A}$ is the number of artist articles. Notice the lack of the coefficient.

Using the extended Bayes' Theorem, the probability that an article containing a given word is a random article is as follows :

$$P(rand|word) = \frac{P(word|rand)P(rand)}{P(word|rand)P(rand) + P(word|\neg rand)P(\neg rand)}$$

Since only random and artist articles exist, we can assume that P(~rand) is P(artist). Furthermore, We can assume that P(rand) = P(~rand), since we said there are an equal amount of artist articles to random articles analyzed, so all instances of P(rand) and P(~rand) can be eliminated from the formula :

$$P(random|word) = \frac{P(word|random)}{P(word|random) + P(word|artist)}$$

Now I need to use the formula for combining conditional probabilities. Instead of using each word in an analyzed article, we will only look at the most "interesting" 15 words, or the 15 words furthest away from 0.5, since this is usually more than enough to identify an article. We do this to remove the effect that mutual words like 'of' or 'the' can have on lowering the proper probability that a given article is a random article.

$$P(rand|words) = \frac{\prod_i^{i=15} P(rand|word_i)}{\prod_i^{i=15} P(rand|word_i) + \prod_i^{i=15} (\neg rand|word_i)}$$

Where “words” is the set of the 15 most interesting words. Since P(~A|B) is just 1-P(A|B), we can rewrite this as :

$$P(rand|words) = \frac{\prod_i^{i=15} P(rand|word_i)}{\prod_i^{i=15} P(rand|word_i) + \prod_i^{i=15} 1 - P(rand|word_i)}$$

I cannot use the formula expressing just P(word|artist) because then we would have to have 4 hash tables : one with the mappings of words onto random amounts, one with the mappings of words onto artist amounts, one with the mappings of words onto P(word|rand), and one with the mappings of words onto P(word|artist), whereas with my method I only have to use 3 hashtables. Each hash table contains a massive amount of words, so this saves on a lot of computational power.

We run this final formula on each article. Usually artist articles will score lower than 1E-8 and random articles score 0.9999999 and above, often rounding to 1.

```
Violinists, a supposed artist article, has a probability of 0.999999883222 of being a random article.
Behzod Abduraimov, a supposed artist article, has a probability of 1.11578359437e-19 of being a random article.
Johann Christian Ludwig Abeille, a supposed artist article, has a probability of 3.88651017299e-05 of being a random article.
Jacques Abram, a supposed artist article, has a probability of 8.27813779984e-08 of being a random article.
Adolovni Acosta, a supposed artist article, has a probability of 9.74768887344e-20 of being a random article.
Daniel Adni, a supposed artist article, has a probability of 1.0 of being a random article.
```

Above : the algorithm locating a non-artist in the database : “Violinists”

```
Coquerel's sifaka, a supposed random article, has a probability of 1.0 of being a random article.
Agapetus montanus, a supposed random article, has a probability of 0.999999999965 of being a random article.
Juvarim, a supposed random article, has a probability of 1.0 of being a random article.
List of presidents of the University of Tulsa, a supposed random article, has a probability of 0.999999988172 of being a random article.
Francs, Gironde, a supposed random article, has a probability of 0.999999999248 of being a random article.
Fairmont Hamilton Princess, a supposed random article, has a probability of 1.0 of being a random article.
```

Above : the algorithm analyzing non-artist articles

There still exists a population of random articles with language very similar to artist articles, such as “Allmusic”, an online music guide. Furthermore, because I only take a subset of genre lists, random articles with a language similar to one of the genre lists often get in. For example, because one of my test runs included numerous video game-related genres (video game composers, video game musicians, etc.), a lot of video game articles like “Star Fox 64” or “Game Boy Color” accidentally fit the language of the

script. Usually, false positives like these don't include many incoming links to the majority of actual artist articles, and thus don't show up often in the final popularity rankings of the program or in the sorted list of artists related to a given artist, and these false positives diminish as the total number of genre lists analyzed increases, diminishing the probability that certain genre lists with certain languages will "attract" non-artist articles with similar language.

Building Incoming and Outgoing Links

To determine artist relatability and popularity, I need to build the weighted directed graph of outgoing and incoming links between artist articles. First, I search through all of the article data for each artist for outgoing links to other artists in our database. For each artist, I build a hash table of outgoing links with artist names mapped onto the amount of time that outgoing link appears on the artist's article.

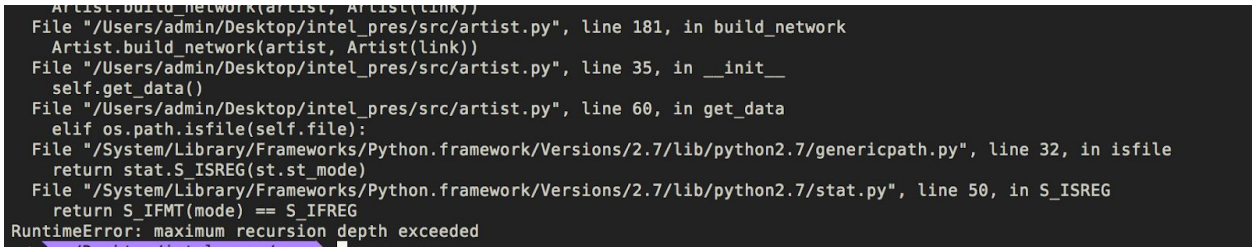
I also build a hash table of incoming links. Every time an outgoing link is built from Artist A to Artist B, I open the file containing the table for Artist B's incoming links and add Artist A mapped onto the amount of B's incoming links from A.

Brief Mention to a Failed Attempt : Building Network Size

A novel approach I've tried to take in determining artist popularity involved assigning a "network score" for each artist. I started by taking an artist's article as a starting node, and then rebuilding the entire database of artists by recursively following incoming links on artists' articles. The network would not add incoming links that already exist (A linking to B who links back to A), and would also stop building articles with no incoming links. After the algorithm has finished building that artist's "network", the ratio between the number of artists in that network to the full number of artists in the database would constitute that artist's network score. Since each artist has different incoming links, there was potential for many of the artists to have different sizes of networks and thus varying network scores.

I follow incoming links instead of outgoing links because it is possible for any artist to edit his/her Wikipedia article with outgoing links to every popular artist, while it is difficult to manipulate incoming links of other popular artists.

Unfortunately, the outcome for this strategy was as expected : every moderately popular artist had an undefined network score. If an artist had even a moderate amount of incoming links, those incoming links had a moderate amount of incoming links, etcetera, etcetera until you hit a moderately sized artist with an incoming link to a very popular artist. Suddenly the network score of moderately popular artists are practically the same as the most popular artists. Every artist that wasn't moderately popular had a tiny amount or no incoming links, making all less-than-moderately-known artists have similar network scores. Because it couldn't distinguish past this binary distinction, this method of determining popularity turned out to be a failure.



Every moderately popular artist had too many incoming links for the script to handle

Determining Popularity : Implementing PageRank

The base algorithm used for building and analyzing the network of artists, PageRank, was initially developed by Google to rank the popularity of websites in their database. PageRank simulates the activities of a hypothetical random web surfer who clicks on random links on web pages; PageRank scores represent the probability that said surfer would end up on any given page. It recursively assigns popularity scores to webpages depending on the volume of incoming links, and the existing popularity score of

incoming links; i.e. to determine the weight of a vertex, we factor in the indegree of that vertex, the weight of the vertices that it is head-incident to, and the outdegree of the vertices it is head-incident to. Intuitively, a vertex 'inherits' its weight from the 'parent' vertices that lead to it, and it inherits an amount proportional to the number of 'children' its parents have.

I will implement PageRank for artist's articles and Wikipedia links as opposed to web pages and hyperlinks.

A preliminary, simplified expression of article *u*'s PageRank is shown :

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Where B_u is the set of all articles that link to *u* and L(*v*) returns the number of outgoing links in *v*. One should note that L(*v*) affects the PageRank of all of *v*'s outgoing links equally. Furthermore, L(*v*) ignores repeats of outgoing links in *v*. For example, if Artist A had 4 outgoing links to page B and 1 outgoing link to page C, L(A) would return 2 in calculating the PageRank for both B and C, affecting the PageRank for both B and C equally despite the fact that A has 4 times more links to B than to C.

In the original paper for PageRank, this is not the final expression of an artist's score. Since PageRank simulates a hypothetical random surfer (or, in our case, someone frantically clicking on Wikipedia links), it accounts for the probability constant "d" that the surfer will get tired and stop clicking. You can express the effect d has on Artist u's PageRank as such :

$$PR(u) = \frac{1-d}{N} + d\left(\sum_{v \in B_u} \frac{PR(v)}{L(v)}\right)$$

Most implementations of PageRank set "d" to 0.85. 'N' is the total amount of artists.

There are many ways to implement PageRank. My algorithm uses a simple iterative recursive method. At t = 0, the PageRank for all artists is :

$$PR(u, t = 0) = \frac{1}{N}$$

Then we increment t and the algorithm iteratively grows :

$$PR(u, t) = \frac{1 - d}{N} + d \left(\sum_{v \in B_u} \frac{PR(v, t - 1)}{L(v)} \right)$$

We can choose the limit of the incrementation of t, depending on how strong our hardware is and how accurate we wish to be.

Here are the top 99 artists, determined by this method, for a short sample of genre lists :

The number 1 artist, Bob Dylan, has a popularity of 0.00289412045133
The number 2 artist, The Rolling Stones, has a popularity of 0.0024897698589
The number 3 artist, The Beatles, has a popularity of 0.00234193800116
The number 4 artist, Michael Jackson, has a popularity of 0.00162690433129
The number 5 artist, Elton John, has a popularity of 0.00159423596048
The number 6 artist, John Lennon, has a popularity of 0.00141371964671
The number 7 artist, Eric Clapton, has a popularity of 0.00139309266351
The number 8 artist, Whitney Houston, has a popularity of 0.0010683220633
The number 9 artist, Robert Johnson (musician), has a popularity of 0.00103546855837
The number 10 artist, Madonna (entertainer), has a popularity of 0.00103073296267
The number 11 artist, Mariah Carey, has a popularity of 0.00101409931728
The number 12 artist, David Bowie, has a popularity of 0.00093619921753
The number 13 artist, Johnny Cash, has a popularity of 0.000919737587497
The number 14 artist, Led Zeppelin, has a popularity of 0.000898326615083
The number 15 artist, George Harrison, has a popularity of 0.000882523456031
The number 16 artist, Ringo Starr, has a popularity of 0.000861203323893
The number 17 artist, Discogs, has a popularity of 0.000848174815428
The number 18 artist, Christina Aguilera, has a popularity of 0.000785451453878
The number 19 artist, Roy Orbison, has a popularity of 0.000783031325159
The number 20 artist, Pink Floyd, has a popularity of 0.000749715783256
The number 21 artist, David Gilmour, has a popularity of 0.000738573465449
The number 22 artist, Mick Jagger, has a popularity of 0.000728834312123
The number 23 artist, Donna Summer, has a popularity of 0.000683731318436
The number 24 artist, Joni Mitchell, has a popularity of 0.00066082674982
The number 25 artist, Roger Waters, has a popularity of 0.000657824564885
The number 26 artist, OverClocked ReMix, has a popularity of 0.000654632513133
The number 27 artist, Arcade Fire, has a popularity of 0.000639432296322
The number 28 artist, Bruce Sudano, has a popularity of 0.000635601658145
The number 29 artist, Jay-Z, has a popularity of 0.000631347285227

The number 30 artist, Yoko Ono, has a popularity of 0.000628235122683
The number 31 artist, The Yardbirds, has a popularity of 0.000624738202602
The number 32 artist, Neil Diamond, has a popularity of 0.000584610094549
The number 33 artist, Woody Guthrie, has a popularity of 0.000582398322231
The number 34 artist, Black Sabbath, has a popularity of 0.000554960829875
The number 35 artist, The Byrds, has a popularity of 0.000551227232938
The number 36 artist, Joan Baez, has a popularity of 0.000549785280275
The number 37 artist, B.B. King, has a popularity of 0.000546625879307
The number 38 artist, Britney Spears, has a popularity of 0.000541617984816
The number 39 artist, Jimmy Page, has a popularity of 0.000528870020242
The number 40 artist, Peter Gabriel, has a popularity of 0.000527058736067
The number 41 artist, Bruce Springsteen, has a popularity of 0.000524717138022
The number 42 artist, Sonic Youth, has a popularity of 0.000524061355126
The number 43 artist, Phil Collins, has a popularity of 0.000513872399567
The number 44 artist, Country Joe McDonald, has a popularity of 0.000510088837677
The number 45 artist, Neil Young, has a popularity of 0.000508945563986
The number 46 artist, Jermaine Jackson, has a popularity of 0.000497337102275
The number 47 artist, Robert Plant, has a popularity of 0.000487468277571
The number 48 artist, Country Joe and the Fish, has a popularity of 0.000477254436985
The number 49 artist, Dolly Parton, has a popularity of 0.00047716231815
The number 50 artist, Stevie Nicks, has a popularity of 0.000473905121416
The number 51 artist, David Crosby, has a popularity of 0.000471503449591
The number 52 artist, Celine Dion, has a popularity of 0.000470499378223
The number 53 artist, Keith Richards, has a popularity of 0.000463763325162
The number 54 artist, Richard Wright (musician), has a popularity of 0.000462960327732
The number 55 artist, Chris Hillman, has a popularity of 0.000458338040235
The number 56 artist, Barbra Streisand, has a popularity of 0.000456715447931
The number 57 artist, Jackson Browne, has a popularity of 0.000455220241099
The number 58 artist, Pete Seeger, has a popularity of 0.00045262354354
The number 59 artist, Radiohead, has a popularity of 0.000452592795151
The number 60 artist, Cream (band), has a popularity of 0.000447095649276
The number 61 artist, Beyoncé, has a popularity of 0.000438706824091
The number 62 artist, Loretta Lynn, has a popularity of 0.000436667633425
The number 63 artist, Kanye West, has a popularity of 0.000436557241152
The number 64 artist, Gibson SG, has a popularity of 0.00043638171784
The number 65 artist, The Who, has a popularity of 0.000423580304144
The number 66 artist, Our Lady Peace, has a popularity of 0.000423333892324
The number 67 artist, Muddy Waters, has a popularity of 0.000422524072153
The number 68 artist, Marianne Faithfull, has a popularity of 0.000417657834746
The number 69 artist, The Band, has a popularity of 0.000414177406841
The number 70 artist, Steve Marriott, has a popularity of 0.000413074534234
The number 71 artist, Lady Gaga, has a popularity of 0.000412119167091
The number 72 artist, C. F. Martin & Company, has a popularity of 0.000411148636653
The number 73 artist, Big Bill Broonzy, has a popularity of 0.000408660222859
The number 74 artist, Annie Lennox, has a popularity of 0.000395319698503
The number 75 artist, Buddy Holly, has a popularity of 0.00039400431644
The number 76 artist, Leon Russell, has a popularity of 0.000391916977332
The number 77 artist, Barenaked Ladies, has a popularity of 0.000391054745588
The number 78 artist, Eminem, has a popularity of 0.000388242445394
The number 79 artist, Alicia Keys, has a popularity of 0.000386970754749
The number 80 artist, David Bromberg, has a popularity of 0.000385820892411
The number 81 artist, Gary Moore, has a popularity of 0.000384315596902
The number 82 artist, ZZ Top, has a popularity of 0.0003840048371
The number 83 artist, Chuck Berry, has a popularity of 0.000380610196309

The number 84 artist, Tom Waits, has a popularity of 0.000379512400649
The number 85 artist, Rodney Crowell, has a popularity of 0.000374877049351
The number 86 artist, Kylie Minogue, has a popularity of 0.000373356971104
The number 87 artist, John Lee Hooker, has a popularity of 0.000368658905184
The number 88 artist, Justin Timberlake, has a popularity of 0.00036649900797
The number 89 artist, Snoop Dogg, has a popularity of 0.000365012372094
The number 90 artist, Chris Murphy (Canadian musician), has a popularity of 0.000364955690774
The number 91 artist, Stephen Stills, has a popularity of 0.000362671339019
The number 92 artist, Stuart Sutcliffe, has a popularity of 0.000361097438738
The number 93 artist, Klaus Voormann, has a popularity of 0.000361097438738
The number 94 artist, Sloan (band), has a popularity of 0.000359966284597
The number 95 artist, James Taylor, has a popularity of 0.000359617228706
The number 96 artist, The Sadies, has a popularity of 0.000357848272858
The number 97 artist, Slash (musician), has a popularity of 0.000356431246772
The number 98 artist, N.W.A, has a popularity of 0.000356149662282
The number 99 artist, Godspeed You! Black Emperor, has a popularity of 0.000355439449747

For reference, here are the 40 genre lists used :

List of Thai pop artists
List of bass guitarists
List of folk metal bands
List of Romanian musicians
Musicians from Denton, Texas
List of Japanoise artists
List of Canadian post-grunge bands
List of South Korean idol groups (1990s)
List of rocksteady musicians
List of musicians who play left-handed
List of Eurobeat artists
List of psychedelic rock artists
List of bands from Finland
List of Israeli musical artists
List of Cape Breton fiddlers
List of Dutch composers
List of jazz blues musicians
List of New Orleans blues musicians
List of Scottish musicians
List of Azerbaijani composers
List of shock rock musicians
List of post-metal bands
List of Los Angeles rappers
List of death metal bands
List of singer-songwriters
Celtic metal#List of bands
List of blues musicians
List of folk rock artists
List of noise musicians
List of house music artists
List of black metal bands
List of Norwegian musicians
List of Las Vegas Valley lounge artists
List of Czech musical groups
List of video game musicians
List of free funk musicians
List of swamp blues musicians
List of deathcore bands
List of bands from Canada
List of electric blues musicians

Conclusion Pt. I : Errors

It's easy to see that the algorithm heavily favors determining popularity in the context of static time. The list is chock full of much older artists like The Rolling Stone, Bob Dylan, The Beatles, Pink Floyd, and Michael Jackson. Many of these artists are no longer active or they are known for their much earlier works. The very first artist that is known for her modern music shows up at 11 with Mariah Carey. While these older artists had a stronger cultural impact on the music industry in their time of activity, these days, there is a larger population of listeners for newer artists. This problem can be easily solved by introducing a weight multiplier in the formula that decreases the value of an artist over the age of their last most popular release. The way it might be computed it as follows :

$$S_f = \frac{S_i}{(t+2)^g}$$

Where $\mathbf{S_i}$ is the initial PageRank score, **t** is the amount of years since the artist's last most popular release, and **g** is the "gravity" constant. The higher the gravity constant, the faster an artist's popularity "decelerates" as it gets older. This is the algorithm Paul Graham uses to rank posts made to Hacker News, his online discussion site. He uses a **g** of 1.5.

In order to find an artist's last most popular release, I would have to scrape the "Discography" section of an artist's article and clean it much as I did for the artist articles themselves. Alternatively, I could use the iTunes Search API to find the artist's best-selling release on iTunes. If neither of those methods work, I could revert to setting **t** to the year in which the artist started being active, a data point much easier to scrape but a method that is slightly more naive.

A second issue involves having a small amount of compact genre articles. This heavily biases the final ratings of relatively unknown artists that are very well-connected in their genre. For example, you

might have noted 26, OverClocked ReMix, ranking higher than many other much better known artists like Arcade Fire, Jay-Z, Britney Spears, and Kanye West. OverClocked ReMix is a video game music collective. I've noted almost every video game-related artist has mentioned OverClocked ReMix. Because OCR is so densely popular in a group of genres, it has a disproportionate amount of incoming links, and thus, it's PageRank is much higher than it should be relative to all other artists. Basically, if each genre list has a similar amount of artists, then each genre list will end up equally represented in terms of PageRank, even though New Orleans blues musicians are far more popular than Azerbaijani composers.

This problem can be solved by increasing the amount of genre articles scraped. By severely increasing the number of genre articles scraped, the value for being heavily popular in one genre article is decreased. This forces popular artists to be well-connected across the board in order to compete.

Also, I could've handled implementing the naive Bayes classifier differently. If you remember, I got rid of all links from poorly titled sections in order to make the assumption that 99% of scraped articles are artist articles and can be used for our artist language without much significant effect, I could've manually found 100 - 200 articles that were actually artist articles, and I could've used those as the basis for our artist 'dictionary' hashtable to train the scraped articles. The problem with this approach is that, with a limited representation of genres, different iterations of the program could give you different artist vocabularies that may or may not vary a lot from our basis artist hashtable. For example, let's say my manually chosen articles had a number of modern musicians, but a certain run of the algorithm randomly chooses generally older genres that have no clue what 'electric guitars' or 'SoundCloud' is, and falsely considers them random words. This is what I have to alleviate by resorting to using the potentially non-perfect scraped corpus as it's own dictionary.

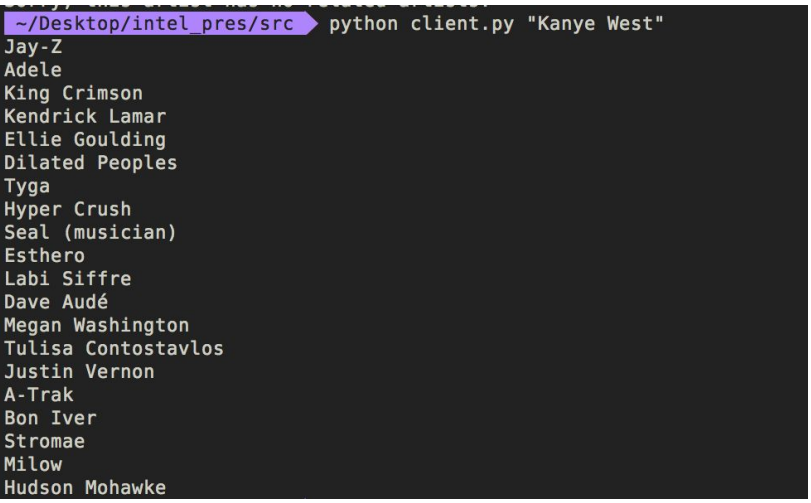
All in all, if I had access to more computational power, I would change the self-defining dictionary method to the manual dictionary training method and have a more representative genre list to more accurately affect artist popularities. Furthermore, I would write up an entirely new section on determining the last time an artist was active, in order to implement the time-gravity feature mentioned earlier.

Conclusion Pt. II : Taking User Input and Recommending Related Artists

If you remember, the original goal of the program was to be able to recommend artists to users. It's not fruitful to merely recommend artists based on pure popularity. The client program will take an artist as input from the user, and output all of the artists from that artists' list of incoming links, sorted by popularity (assuming artists that are popular with other artists will also be popular with users). Here are 2 sample inputs.

```
~/Desktop/intel_pres/src > python client.py "The Beatles"
Bob Dylan
Jimmy Page
Phil Collins
Keith Richards
Jackson Browne
Lady Gaga
Barenaked Ladies
Eminem
Stuart Sutcliffe
Klaus Voormann
James Taylor
Nazareth (band)
The Great Society (band)
Bono
Buck Owens
Liam Gallagher
Terry McDermott (singer)
Andy White (drummer)
Kurt Cobain
Rush (band)
Nickelback
Louis Armstrong
Nick Cave
The Incredible String Band
Broken Records (band)
Dave Grohl
Chris Squire
Branimir Štulić
Roberta Flack
Billy Joel
Maurice Gibb
Rico Blanco
Lemmy
Gene Simmons
Michelle Branch
Jefferson Airplane
Matt Bissonette
Robert Smith (musician)
The Verve
John Myung
Bay City Rollers
Can (band)
Elmore James
Neil Sedaka
Drake Bell
Ohbijou
Lonnie Donegan
Vanilla Fudge
Seal (musician)
Os Mutantes
```

Going from the bottom-up, we see the program's usefulness in actually recommending more experimental, less popular artists from similar genres and eras - Nick Cave, Can, Robert Smith, Jefferson Airplane, etc.



Includes other popular rappers like Jay-Z, Kendrick Lamar, and Tyga, as well as Kanye collaborators and artists that Kanye has sampled (King Crimson)

Reference 1 :

http://en.wikipedia.org/wiki/Music_Genome_Project

“Given the vector of one or more songs, a list of other similar songs is constructed using what the company calls its ‘matching algorithm’. Each song is analyzed by a musician in a process that takes 20 to 30 minutes per song.”

Reference 2: http://notes.variogr.am/post/37675885491/how-music-recommendation-works-and-doesnt-work

“Even websites that can be volunteer or community edited run against the limits of the community that takes part – we count only a little over 130,000 artist pages on Wikipedia.”

Reference 3 :

https://mail.python.org/pipermail/python-dev/2002-August/028216.html

Reference 4 :

http://en.wikipedia.org/wiki/Lists_of_musicians

Reference 5 :

http://docs.python-requests.org/en/latest/

Reference 6:

http://www.mediawiki.org/wiki/API:Main_page

Reference 7:

http://en.wikipedia.org/wiki/PageRank

“Various studies have tested different damping factors, but it is generally assumed that the damping factor will be set around 0.85”

Reference 8 :

https://news.ycombinator.com/item?id=231209